

1 Homework Exercises

Reading: From text, Chapter 1, section 1.2 and 1.3.

Write up and turn in the following exercises:

1. Do exercise 1.9 in the textbook.

```
(define (+ a b)
  (if (= a 0)
      b
      (inc (+ (dec a) b))))
```

```
(+ 4 5)
```

Solution:

```
(inc (+ 3 5))
(inc (inc (+ 2 5)))
(inc (inc (inc (+ 1 5))))
(inc (inc (inc (inc (+ 0 5)))))
(inc (inc (inc (inc 5))))
(inc (inc (inc 6)))
(inc (inc 7))
(inc 8)
;value 9
```

This process is recursive.

Part 2:

```
(define (+ a b)
  (if (= a 0)
      b
      (+ (dec a) (inc b))))
```

```
(+ 4 5)
```

Solution:

```
(+ 3 6)
(+ 2 7)
(+ 1 8)
(+ 0 9)
;value 9
```

This process is iterative.

2. Do exercise 1.11 in the textbook.

```
(define (fn-recurs n)
  (if (< n 3)
      n
      (+ (fn-recurs (- n 1)) (* 2 (fn-recurs (- n 2))) (* 3
      (fn-recurs (- n 3))))))
```

```

(define (f n)
  (define (f-iter a b c count max)
    (if (> count max)
        c
        (f-iter b c (+ (* 3 a) (* 2 b) c) (1+ count) max)))
  (if (< n 3)
      n
      (f-iter 0 1 2 3 n)))

```

3. Do exercise 1.16 in the textbook.

```

(define (square x) (* x x))
(define (even? n) (= (remainder n 2) 0))
(define (func b n) (iter 1 b n))
(define (iter a b n)
  (cond ((= n 0) 1)
        ((= n 1) (* a b))
        ((even? n) (iter a (square b) (/ n 2)))
        (else (iter (* a b) b (- n 1)))))

```

4. Do exercise 1.19 in the textbook.

```

(define (fib n)
  (fib-iter 1 0 0 1 n))

(define (fib-iter a b p q count)
  (cond ((= count 0) b)
        ((even? count)
         (fib-iter a
                   b
                   (+ (* p p) (* q q))
                   (+ (* q q) (* 2 q p))
                   (/ count 2))))
  (else (fib-iter (+ (* b q) (* a q) (* a p))
                  (+ (* b p) (* a q))
                  p
                  q
                  (- count 1)))))

```

5. Do exercise 1.26 Louis's procedure `expmod` is in $O(n)$ time because he calls `expmod` twice to do the multiplication instead of calling `expmod` once and squaring it.

6. Do exercise 1.30 in the textbook.

```

(define (sum-iter term a next b)
  (define (iter a result)
    (if (> a b)
        result
        (iter (next a) (+ result (term a)))))
  (iter a 0))

```

7. Do exercise 1.32 in the textbook.

```
(define (accumulate combiner null_value term a next b)
  (if (> a b)
      null_value
      (combiner (term a) (accumulate combiner null_value term (next a) next b))))
```

```
(define (accumulate combiner null_value term a next b)
  (define (iter a result)
    (if (> a b)
        result
        (iter (next a) (combiner result (term a)))))
  (iter a null_value))
```

8. Do exercise 1.41 in the textbook.

```
(define (inc n)
  (+ 1 n))
;Value: inc
```

```
(define (double fn)
  (lambda (x)
    (fn (fn x))))
;Value: double
```

```
((double inc) 5)
;Value: 7
```

```
((double double) inc) 5)
;Value: 9
```

```
((double (double double)) inc) 5)
;Value: 21
```

9. Exercise 5.1 They are the same except for the syntactic sugar.

10. Exercise 5.2 If every word was encrypted separately, there would only be the problem of breaking the private key and it could be done on any part of the message since the encryption only depends on the private key. This would be easier to crack by finding a pattern than only having a unique relationship (the message and the private key) on just one element of the message.

11. Exercise 5.3 You should encrypt and then sign the messages. The reason is that if you apply your signature on each word's code and then encrypt them you have introduced a common number to each word which will make it easier to find a pattern. Also, if you only send the compressed smaller number it would be increase your chances of having it cracked since the smaller the number the easier it is.